Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЕТ

к лабораторной работе № 6

на тему

**ЭЛЕМЕНТЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ**

Выполнил:

студент гр. 153503

Кончик Д.С.

Проверил:

Гриценко Н.Ю.

Минск 2024

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 5](#_Toc146836469)

[Выводы 6](#_Toc146836470)

[Список использованных источников 7](#_Toc146836471)

[Приложение А (обязательное) листинг кода 8](#_Toc146836472)

# **1 ЦЕЛЬ РАБОТЫ**

Практическое освоение основ построения и функционирования сетей, стеков протоколов, программных интерфейсов. Изучение сетевой подсистемы и программного интерфейса сокетов в Unix-системах. Практическое проектирование, реализация и отладка программ, взаимодействующих через сеть TCP/IP.

Написать программу, реализующую упрощенный чат для нескольких пользователей с использованием сетевых сокетов. Сервер должен создать сокет для приема соединений или отдельных сообщений, передавать сообщения адресно одному или нескольким клиентам. Клиент должен обнаруживать сервер, отправлять пользовательские сообщения серверу.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ** 

Сетевое программирование – это область программирования, связанная с разработкой приложений, способных взаимодействовать и обмениваться данными через компьютерные сети. Эта область охватывает создание программ, работающих в распределенной среде, где различные компьютеры могут общаться между собой, отправлять и принимать данные.

Сеть представляет собой инфраструктуру, позволяющую различным устройствам обмениваться информацией. Это может быть локальная сеть (*LAN*) в пределах одного офиса или дома, глобальная сеть, такая как Интернет, или виртуальная сеть между программами на одном компьютере.

Сокет – это виртуальная конструкция из *IP*-адреса и номера порта. Сокет служит для того, чтобы было проще писать код, а программы могли передавать данные друг другу даже в пределах одного компьютера. Т.к. сокет на сервере один, а программ, которые должны подключаться, много, то сервер копирует сокеты. Когда на сервер поступает запрос на соединение с сокетом, он не устанавливает связь напрямую, а копирует этот сокет и настраивает связь через него. После копирования сервер запоминает, какая копия отвечает за какое соединение, и дальше просто обрабатывает все запросы по очереди. При этом исходный сокет остаётся нетронутым – он не используется для связи напрямую, а служит шаблоном для создания копий [1].

Протоколы представляют собой набор правил и соглашений, определяющих формат, порядок и правила взаимодействия между устройствами или программами в сети. Они служат основой для обмена данными и обеспечивают структурированный способ передачи информации между различными компьютерами, устройствами или системами. Протоколы могут работать на разных уровнях стека сетевых протоколов (например, модель *OSI* или *TCP/IP*), предоставляя структуру для обработки различных аспектов передачи данных.

Сетевая модель *OSI* – сетевая модель стека сетевых протоколов *OSI/ISO*. Посредством данной модели различные сетевые устройства могут взаимодействовать друг с другом. Модель определяет различные уровни взаимодействия систем. Каждый уровень выполняет определённые функции при таком взаимодействии [2].

*TCP/IP* – это модель передачи цифровых данных. Протокол передачи *TCP/IP* описывает правила передачи данных, стандарты связи между компьютерами, а также содержит соглашения о маршрутизации и межсетевом взаимодействии [3].

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

В результате выполнения лабораторной работы был написана программа, реализующая Упрощенный чат для нескольких пользователей с использованием сетевых сокетов.

Запускается сначала сервер, потом к нему подключаются клиенты. Клиенты могут отправлять сообщения на сервер. Сервер эти сообщения отправляет всем клиентам, за исключением того, от кого оно пришло. Сервер фиксирует подключение и отключение клиентов(рисунок 1).

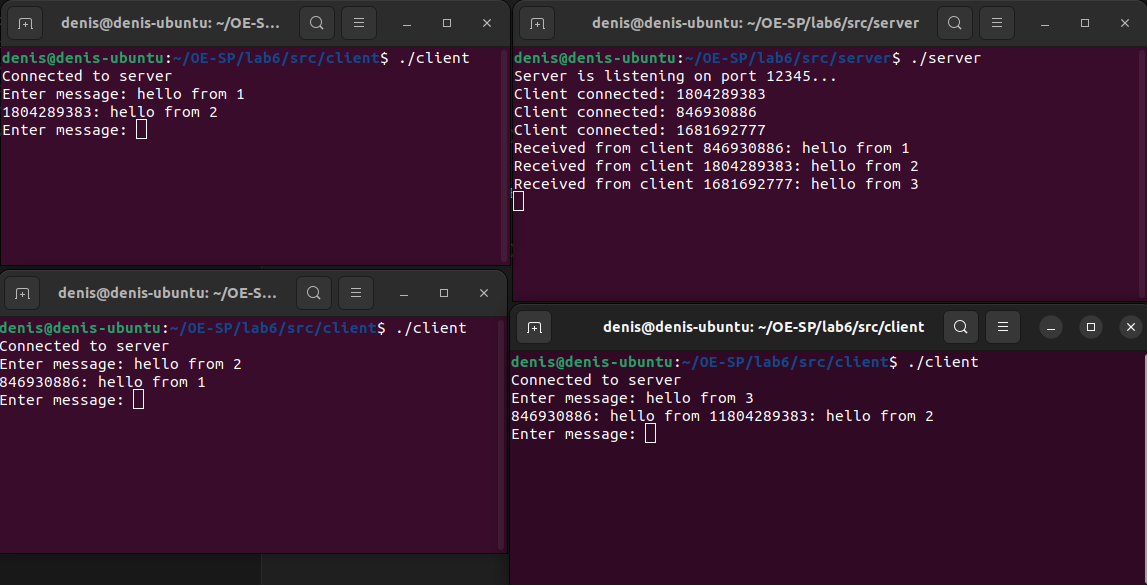


Рисунок 1 – Результат работы программы

# **ВЫВОДЫ**

В результате выполнения лабораторной работы были практически освоены основы построения и функционирования сетей, стеков протоколов, программных интерфейсов. Изучены сетевые подсистемы и программные интерфейсы сокетов в Unix-системах.

Написана программа, реализующая упрощенный чат для нескольких пользователей с использованием сетевых сокетов. Сервер создает сокет для приема соединений или отдельных сообщений, передает сообщения адресно одному или нескольким клиентам. Клиент обнаруживает сервер, отправляет пользовательские сообщения серверу.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Сокет [Электронный ресурс]. – Режим доступа: https://thecode.media/socket/.
2. Сетевая модель *OSI* [Электронный ресурс]. – Режим доступа: https://ru.wikipedia.org/wiki/Клиент\_–\_сервер.
3. *TCP/IP* [Электронный ресурс]. – Режим доступа: https://www.nic.ru/help/chto-takoe-tcpip\_11168.html.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *server.c*:

*#include* <stdio.h>  
*#include* <stdlib.h>  
*#include* <string.h>  
*#include* <unis  
*#include* <sys/socket.h>  
*#include* <netinet/in.h>  
*#include* <pthread.h>  
  
*#define* DEFAULT\_PORT 12345  
*#define* BUFFER\_SIZE 2056  
*#define* MAX\_CLIENTS 10  
  
*struct* ClientInfo {  
 *int* socket;  
 *char* clientId[20];  
};  
  
*struct* ClientInfo clients[MAX\_CLIENTS];  
*int* num\_clients = 0;  
  
*char*\* generate\_client\_id() {  
 *char*\* client\_id = malloc(20 \* *sizeof*(*char*));  
 *if* (client\_id == NULL) {  
 perror("Memory allocation failed");  
 exit(EXIT\_FAILURE);  
 }  
  
 snprintf(client\_id, 20, "%d", rand());  
 *return* client\_id;  
}  
  
*void*\* handle\_client(*void*\* client\_info\_ptr) {  
 *struct* ClientInfo\* client\_info = (*struct* ClientInfo\*)client\_info\_ptr;  
 *int* client\_socket = client\_info->socket;  
 *char* buffer[BUFFER\_SIZE];  
  
 *while* (1) {  
 *int* bytes\_received = recv(client\_socket, buffer, *sizeof*(buffer) - 1, 0);  
 *if* (bytes\_received == -1 || bytes\_received == 0) {  
 printf("Client %s disconnected\n", client\_info->clientId);  
 close(client\_socket);  
 *return* NULL;  
 }  
  
 buffer[bytes\_received] = '\0';  
 printf("Received from client %s: %s\n", client\_info->clientId, buffer);  
  
 *for* (*int* i = 0; i < num\_clients; ++i) {  
 *if* (clients[i].socket != client\_socket) {  
 *char* message\_with\_id[BUFFER\_SIZE];  
 snprintf(message\_with\_id, *sizeof*(message\_with\_id), "%s: %s", client\_info->clientId, buffer);  
 send(clients[i].socket, message\_with\_id, strlen(message\_with\_id), 0);  
 }  
 }  
 }  
}  
  
*int* main() {  
 *struct* sockaddr\_in server\_addr, client\_addr;  
 socklen\_t client\_addr\_len = *sizeof*(client\_addr);  
 pthread\_t client\_threads[MAX\_CLIENTS];  
  
 *int* server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
 *if* (server\_socket == -1) {  
 perror("Failed to create server socket");  
 exit(EXIT\_FAILURE);  
 }  
  
 server\_addr.sin\_family = AF\_INET;  
 server\_addr.sin\_addr.s\_addr = INADDR\_ANY;  
 server\_addr.sin\_port = htons(DEFAULT\_PORT);  
  
 *if* (bind(server\_socket, (*struct* sockaddr\*)&server\_addr, *sizeof*(server\_addr)) == -1) {  
 perror("Failed to bind server socket");  
 close(server\_socket);  
 exit(EXIT\_FAILURE);  
 }  
  
 *if* (listen(server\_socket, SOMAXCONN) == -1) {  
 perror("Error in listen function");  
 close(server\_socket);  
 exit(EXIT\_FAILURE);  
 }  
  
 printf("Server is listening on port %d...\n", DEFAULT\_PORT);  
  
 *while* (1) {  
 *int* client\_socket = accept(server\_socket, (*struct* sockaddr\*)&client\_addr, &client\_addr\_len);  
 *if* (client\_socket == -1) {  
 perror("Failed to accept client connection");  
 *continue*;  
 }  
  
 *if* (num\_clients >= MAX\_CLIENTS) {  
 printf("Maximum number of clients reached. Closing new connection.\n");  
 close(client\_socket);  
 *continue*;  
 }  
  
 *struct* ClientInfo new\_client;  
 new\_client.socket = client\_socket;  
 strncpy(new\_client.clientId, generate\_client\_id(), *sizeof*(new\_client.clientId));  
  
 printf("Client connected: %s\n", new\_client.clientId);  
 clients[num\_clients++] = new\_client;  
 pthread\_create(&client\_threads[num\_clients - 1], NULL, handle\_client, (*void*\*)&clients[num\_clients - 1]);  
 }  
}

Листинг 2 **–** Файл *client.c*:

*#include* <stdio.h>  
*#include* <stdlib.h>  
*#include* <string.h>  
*#include* <unistd.h>  
*#include* <arpa/inet.h>  
*#include* <sys/socket.h>  
*#define* SERVER\_IP "127.0.0.1"  
*#define* SERVER\_PORT 12345  
*#define* BUFFER\_SIZE 1024  
  
*int* main() {  
 *struct* sockaddr\_in server\_addr;  
 *int* client\_socket;  
   
 client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
 *if* (client\_socket == -1) {  
 perror("Socket creation failed");  
 exit(EXIT\_FAILURE);  
 }  
   
 server\_addr.sin\_family = AF\_INET;  
 server\_addr.sin\_port = htons(SERVER\_PORT);  
 *if* (inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr) != 1) {  
 perror("Invalid address");  
 close(client\_socket);  
 exit(EXIT\_FAILURE);  
 }  
   
 *if* (connect(client\_socket, (*struct* sockaddr \*)&server\_addr, *sizeof*(server\_addr)) == -1) {  
 perror("Connection failed");  
 close(client\_socket);  
 exit(EXIT\_FAILURE);  
 }  
 printf("Connected to server\n");  
 *char* buffer[BUFFER\_SIZE];  
 *while* (1) {  
 printf("Enter message: ");  
 fgets(buffer, BUFFER\_SIZE, stdin);  
 buffer[strcspn(buffer, "\n")] = 0;   
   
 *if* (send(client\_socket, buffer, strlen(buffer), 0) == -1) {  
 perror("Message sending failed");  
 *break*;  
 }  
 *int* bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE - 1, 0);  
 *if* (bytes\_received <= 0) {  
 perror("Server disconnected");  
 *break*;  
 }  
 buffer[bytes\_received] = '\0';  
 printf("%s\n", buffer);  
 }  
 close(client\_socket);  
  
 *return* 0;  
}